Virtual Grid Symbolic Layout

Neil Weste

Bell Laboratories
Holmdel, New Jersey 07733

ABSTRACT

Free form or "stick" type symbolic layout provides a means of simplifying the design of IC subcircuits. To successfully utilize this style of layout, a complete design approach and the necessary tools to support this methodology are required. In particular, one of the requirements of such a design method is the ability to "compact" the loosely specified topology to create a set of valid mask data. This paper presents a new compaction strategy which uses the concept of a *virtual grid*. The compaction algorithm using the virtual grid is both simple and fast, an attribute which allows the designer to conveniently interact with the algorithm to optimize a layout. In addition to the compaction algorithm, methods used to create large building blocks will be described. The work described here is part of a complete symbolic layout system called MULGA which is written in the C programming language and resides on the UNIX operating system.

1. INTRODUCTION

A proven method of designing integrated circuits is by reducing functions to manageable subcircuits, implementing these subcircuits, verifying their performance and then using automatic or structured techniques to build larger subcircuits until the complete chip is specified. Based on this design philosophy, there is a need for systems which allow the design of cells in an error free manner and the subsequent documentation of their performance. One method for reducing the complexity of the physical design of cells is to use symbolic layout techniques. These methods reduce or completely eliminate the many geometric design rules generally needed to specify an IC mask set, and provide additional benefits at the *capture* stage of a design.

This paper describes the compaction strategies that are used in MULGA [1], a UNIX-based interactive symbolic layout system. The components used in this task consist of a compaction program and various methods of converting the symbolic description into a valid mask set. All software is written in the C programming language and runs under the UNIX operating system on a microcomputer (DEC PDP 11/23) based design station.

2. MULGA

Symbolic layout methodologies use symbolism to reduce the complexity of custom mask design. Apart from offering densities comparable to manually designed mask layouts, symbolic layout methods provide potentially shorter design times, a reduced liklihood of errors and a technology tracking library. MULGA is a UNIX-based interactive symbolic design system consisting of a suite of programs residing on a high performance color display station. The MULGA system has been designed to provide the smooth man-machine dialogue necessary to support the tasks required during design capture and performance verification of IC subcircuits. Figure 1 summarises the software modules that constitute MULGA and the paths along which information is passed to complete a design.

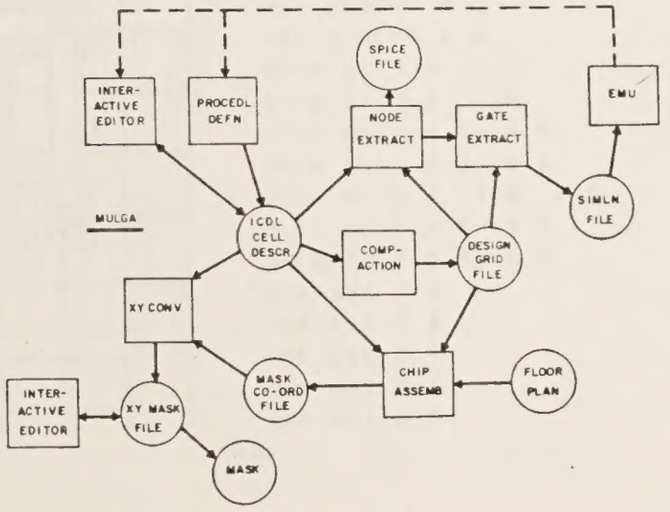


Figure 1.

Software Modules in the MULGA Design System

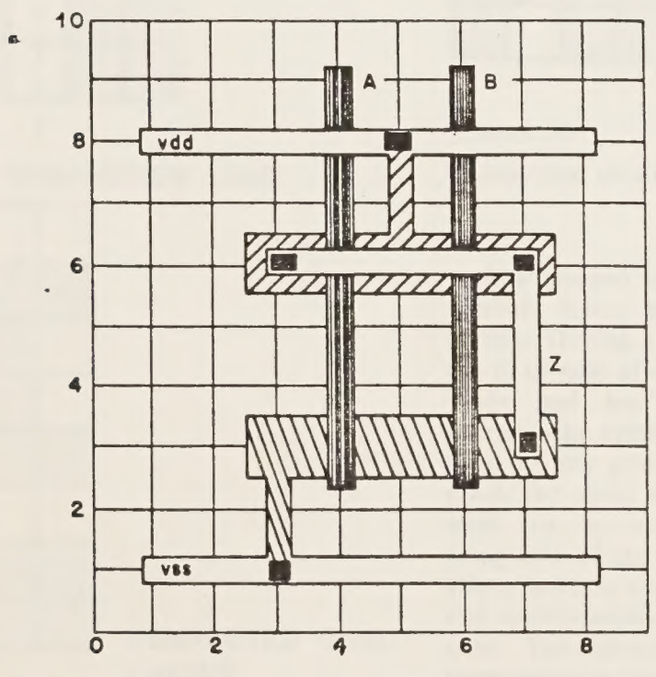
• UNIX is a Bell System Operating System

0146 7123/81/0000-0225$00.75 © 1981 IEEE

The system uses as a central data base, a symbolic Intermediate Circuit Description Language (ICDL), which uses a derivation of the *coordinode* notation used by Buchanan [2]. The language combines the physical and structural aspects of a design by using data elements which have predefined geometric properties in conjunction with designated connection points which are used to establish electrical connectivity. In this way the language aids the capture of designer intent with respect to circuit topology and geometric placement.

The basic structure in ICDL is a *cell* which consists of a variety of elements placed in the X-Y plane as shown in Figure 2. These elements may be **devices** (transistors), **wires, contacts, pins** or other cell **instances.** Devices are the active elements in a process. Wires, which may exist on any interconnection layer, serve to interconnect devices in conjunction with contacts. Pins are used to name internal nodes in a circuit or to specify connection points on the boundary of a cell. Although they have no physical significance in the final mask representation of the circuit, they are an important attribute of the language being used in cell placement and circuit verification. Cell instances allow cells to be defined hierarchically and in addition allow arrays to be easily specified. Figure 2 depicts a CMOS 2-input nand gate in a graphical representation, with the ICDL textual description for that cell. It is evident from this diagram that each line of text represents a circuit element rather than a geometrical shape as in conventional physical mask description languages. Each component is snapped to a grid as shown. Note, however, that this grid does not initially have any physical mask related spacing. Subsequent processing of the ICDL description yields the spacing between grid points. The attribute of having a topology driven spacing for the grid is thought of as a *virtual grid* which is the name given to this type of layout. This paper will deal with the methods of processing the ICDL to yield a valid mask set.

Before dealing with the compaction methods, the overall design methodology used in MULGA will be reviewed. ICDL cell descriptions may be generated interactively via a color graphics/text editor or alternatively, procedurally defined using the C programming language. Once physically designed, the cell may be checked structurally using a circuit extractor and a gate extractor [3], which produce text descriptions suitable for a circuit analysis program or timing simulator, respectively. The timing simulator EMU [3] runs on the design terminal and provides initial estimates of timing performance. When the cell has been verified behaviourally, the final mask description is generated using a *chip-assembly* program, which uses as guidance, a chip floor-plan.




```
begin nand2
   pin al 1 1 vss
   pin al 1 8 vdd
   pin poly 4 9 A
   pin poly 6 9 B
   pin al 7 5 Z
   dev n or=1 4 3
   dev n or=1 6 3
   dev p or=1 4 6
   dev p or=1 6 6
   wire al 1 1  8 1
   wire al 1 8  8 8
   wire poly 4 3  4 9
   wire poly 6 3  6 9
   wire al 3 6  7 6  7 3
   wire Ndif 3 1  3 3
   wire Pdif 5 6  5 8
   con cut 7 3
   con cut 7 6
   con cut 3 1
   con cut 3 6
   con cut 5 8
end
```

Figure 2. A 2-input Nand gate
- graphical representation
- ICDL description

3. COMPACTION

3.1 Introduction

Once a topology has been generated, it is necessary to *compact* the design to produce a mask description. One of the first examples of a compaction algorithm was by Akers et.al.[4]. This scheme was based on a fixed-grid system where the layout was represented as entries in a rectangular matrix. Each entry in the matrix represented a fixed sized mask element. The matrix thus constructed was searched to determine a path of blank cells across the matrix. Paths were allowed to be straight lines as shown in Figure 3a or could follow what were termed *shear lines* as shown in Figure 3b. Rules applied to paths to protect the topology when space was extracted from the matrix. Compaction proceeded by repeatedly scanning in the X then Y directions until no paths could be found or some other terminating condition was satisfied.

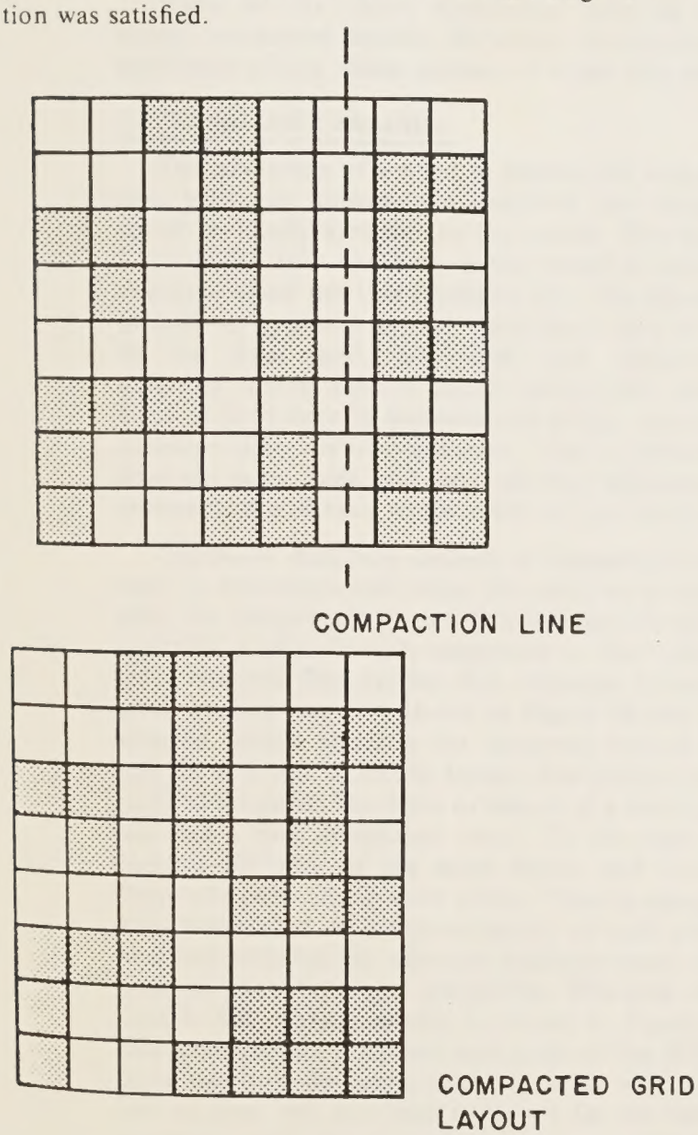


Figure 3a.

Compaction on a Coarse Grid - Simple Example

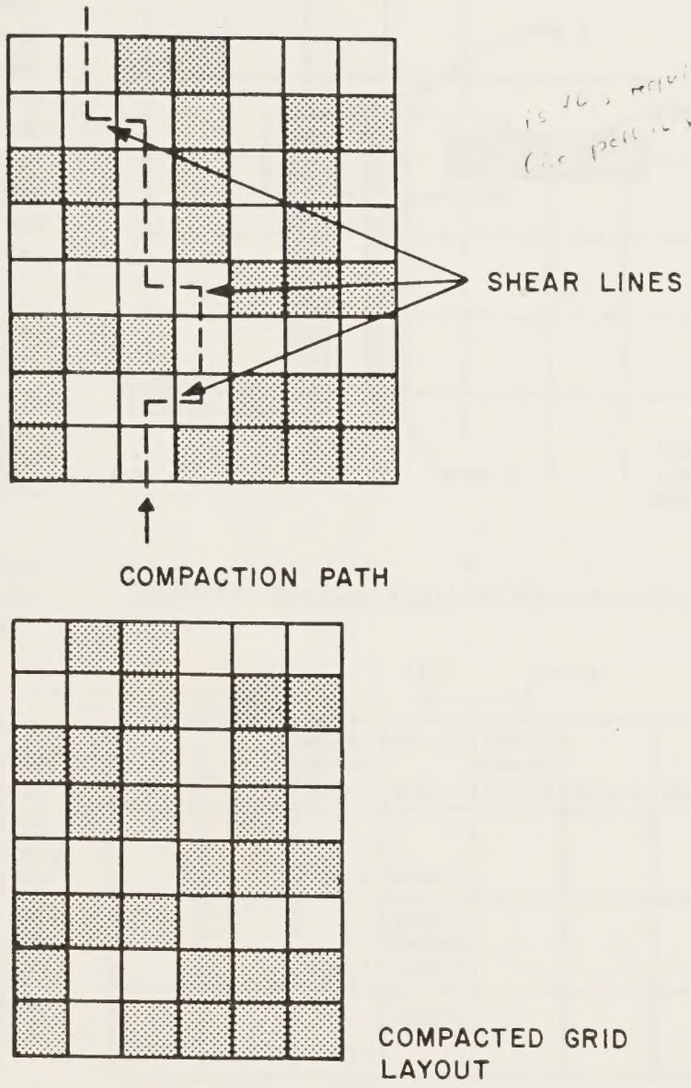


Figure 3b.

Compaction on a Coarse Grid - Use of Shear Lines

The general idea of compacting a "sticks" type of symbolic layout was first described in [5]. Subsequent to this, Dunlop [6] proposed a detailed scheme where the fixed grid of a coarse grid compactor is traded for a "node and line" symbolic representation. In this scheme, the symbols existed as objects which were not fixed to any grid. These objects were manipulated in much the same way as design rule checking programs work (i.e. by comparing element to element spacings using lists of the elements) to arrive at a set of allowable geometric spacings between objects. The same general search techniques as used in Akers' algorithm were used. The advantage of using the "node and line" representation was potentially improved packing density at the expense of execution time.

A compaction algorithm which used a graph theory approach was proposed by Hseuh et.al. [7]. In this algorithm the symbolic layout was partitioned into vertically

connected features and horizontally connected features. Features on a common center line were then grouped to prevent the compaction process separating them. Spacing relationships between such collections of features were then used to construct a graph which was weighted according to design rules between features. The most costly path was then determined and used to derive final mask coordinates. Again this was repeated in the X and Y directions until some termination condition was satisfied.

A combination of this compactor and the "node and line" compactor was used in a system called SLIM[8]. Compaction was divided into "local" and "global" phases. The minimum cost graph compactor was used locally and the "node and line " globally. Other refinements such as jog insertion and automatic routing were used to alter the topology to extract space.

While all the above approaches have all demonstrated compacted layouts, the author is unaware of the application of any these systems to actual chip designs.

3.2 Virtual Grid Compaction

The advantage of a fixed or coarse grid compactor is that adjacency information required for design rule checks is locally available in the matrix. This results in rapid design rule checking, a fact noted in design rule checkers which use this approach [9]. The disadvantage of such an approach is that mask space may be wasted by the fixed grid. Non-fixed grid compactors as described above achieve better compaction but spend most of their time in conventional design rule type calculations (i.e. list comparisons). Thus a system which does not use a fixed grid but does treat adjacency information on a grid basis would seem to have merit.

The ICDL data base consists of elements or "objects" such as transistors and wires that exist on a conceptual grid. For instance, in Figure 4a a transistor is shown (in symbolic graphic format) connected to four wires using three contacts. Because the data structure is based on a grid, the data structure shown in Figure 4b may be constructed, which indicates the elements located at each grid point in the symbolic layout. For instance at point [2,3], there exists the drain or source of a device, a contact and a wire designated *wire1*. To the right of this point is the gate of the same device and wire *wire2*. Below the point [2,3], *wire1* exists. Thus by using a suitably constructed data representation at each grid point one may examine the adjacent neighbourhood of a grid point to determine local constraints. The data structure used in the current system is shown in Figure 4c. At each grid location a pointer may point to the ICDL data structures for a device, a contact and two wires. In practice, to date, this has been sufficient for the fabrication processes used, but may of course be modified to suit different technologies (i.e. two-level metal). Considering compaction in the X direction firstly then, the matrix shown in Figure 4b is examined column-wise. Thus, when travelling up column 2 and looking to the

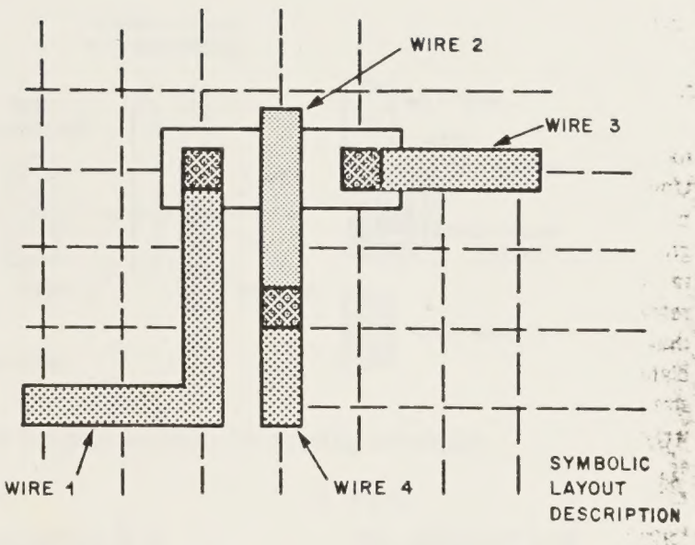


Figure 4a. Symbolic Graphical Representation

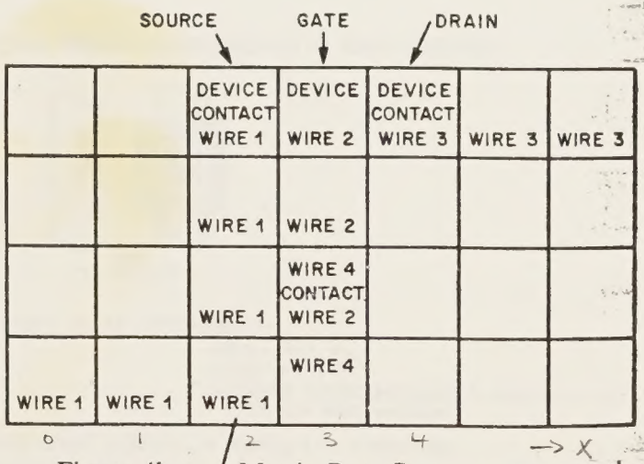


Figure 4b. Matrix Data Structure representing 4a

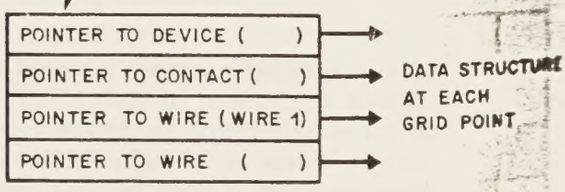


Figure 4c. The data structure at each matrix element

right, *wire4*, *wire2*, a contact and a device would be found adjacent to *wire1*.

Thus, the first step in compaction is to plot the ICDL description into the array using the data structure shown in Figure 4c. On small machines only two columns or rows need be plotted at once, thus conserving memory space. Wires are regarded as stretchable in the direction of compaction. To achieve this, the breakpoints (or bends) in a wire are plotted when compacting in both directions, but only interior elements on wires

perpendicular to the direction of compaction are plotted. For instance, Figure 5a shows the actual data structure used for the X compaction of Figure 4a. Note that grid point [1,0] is not plotted as part of *wire1*. Similarly point [5,3] of *wire3* is not plotted. The data structure for Y compaction is shown in Figure 5b. Here grid points in the vertical sections of *wire1* and *wire4* have been omitted. This procedure ensures that wires collapse to a minimum length if topology permits. Apart from wires, the only other element treated specially is the device (or transistor). A device is plotted as three points, representing the drain, gate and source. Contacts are plotted as single entries on the grid.

		DEVICE CONTACT WIRE 1	DEVICE WIRE 2	DEVICE CONTACT WIRE 3		WIRE 3
		WIRE 1	WIRE 2			
		WIRE 1	WIRE 4 CONTACT WIRE 2			
WIRE 1		WIRE 1	WIRE 4			

Figure 5a. Actual matrix during X compaction

		DEVICE CONTACT WIRE 1	DEVICE WIRE 2	DEVICE CONTACT WIRE 3	WIRE 3	WIRE 3
			WIRE 4 CONTACT WIRE 2			
WIRE 1	WIRE 1	WIRE 1	WIRE 4			

Figure 5b. Matrix during Y compaction

When the circuit has been plotted into the array, adjacent columns are examined pairwise from left to right and from bottom to top for non-zero entries. Upon finding two adjacent points which reference non-zero elements, the relative mask dimension of each point is then calculated. This is achieved by examining the elements and applying a procedure to each element to find its contribution to the dimension on each available mask level. For instance Figure 6a shows in symbolic graphic representation a vertical polysilicon wire

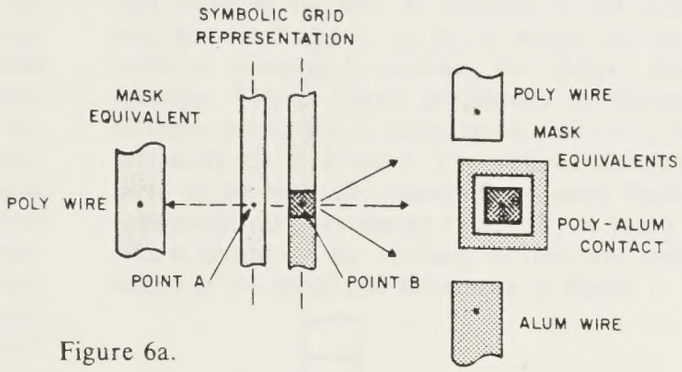


Figure 6a.

Symbolic Graphic example for spacing calculation

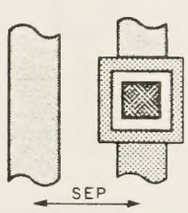
"A" WIDTHS

	DEVICE	CONTACT	WIRE
POLY	0	0	4
ALUM	0	0	0
DIFF	0	0	0

"B" WIDTHS

	DEVICE	CONTACT	WIRE
POLY	0	5	4
ALUM	0	7	4
DIFF	0	0	0

Figure 6b. Width contributions of each element



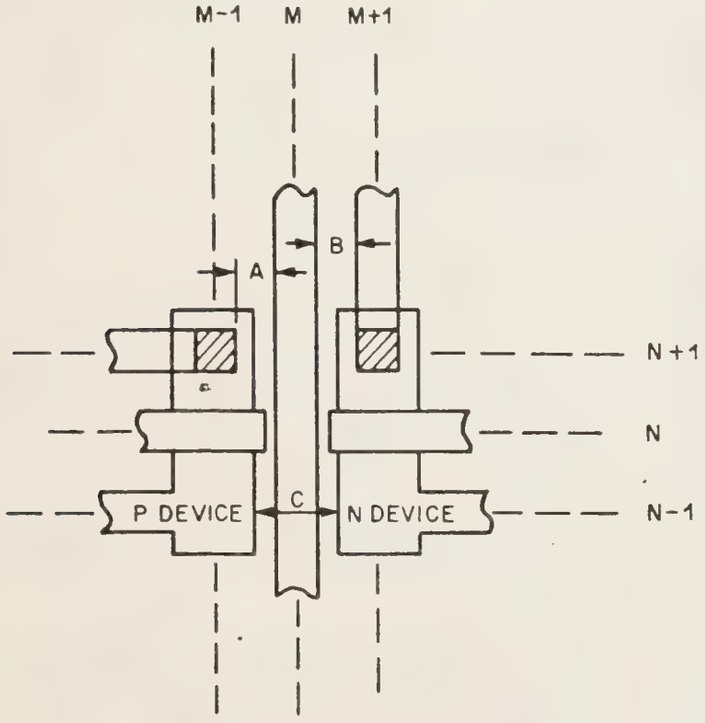
SPACING DESIGN RULES APPLICABLE :-
POLY - POLY 4
ALUM - POLY 0
(WINDOW RULES ACCOUNTED FOR BY CONTACT CONSTRUCTION)

$SEP = \text{WORST CASE } ((WIDTH_A + WIDTH_B)/2 + SPACING)$
$= ((4+5)/2+4)$
$= 8.5 \text{ UNITS}$

Figure 6c. Spacing calculation

adjacent to a contact joining a vertical polysilicon wire to an aluminum wire. The points of interest, A and B, have been designated. Considering compaction in the X direction, the mask contribution at A would be the width of the polysilicon wire. At point B, there are three contributions, due to the contact, aluminum wire and polysilicon wire. A table may be constructed for the contributions to the "width" of each point from each coincident element. Figure 6b summarises these for both points. At point A there is a single polysilicon contribution from the wire, while at B there are aluminum and polysilicon contributions from the contact and the two wires. To calculate the separation required between these two points, the allowed interlayer spacings for the process are combined with the worst case widths on each layer to arrive at a design rule correct value. Figure 6c summarises this calculation, indicating for the widths and spacings shown, these two grid points could be at minimum, 8.5 units apart. Normally, mask contribu-

tions are viewed as symmetric around the axes of the grid but this need not be the case. The maximum contribution on each mask level due to any element is used to ensure correct spacing. In addition to the dimensions of the adjacent grid points, the connectivity of the elements is checked and the spacing modified accordingly. The worst case spacing for comparing the two columns for the height of the cell is then used as the notional minimum mask spacing between these two grid values. Apart from checking strictly adjacent column entries, the program backtracks to the left to check previous entries. In this manner design rules which exert their influence over many grid units are incorporated. Figure 7 shows this effect. By obeying a strict adjacency scheme the critical spacings between column m-1,m and m,m+1 would appear to be spacings A and B due to aluminum spacing. In fact the correct spacing is the P-device to N-device spacing shown at C. As columns are scanned from left to right (in the case of X compaction) the row position of the worst case spacing elements is stored for future reference.



STRICT ADJACENCY WOULD INDICATE SPACINGS A AND B ARE DOMINANT
HOWEVER SPACING C – N DIFFUSION TO P DIFFUSION IS DOMINANT

Figure 7.

Backtracking to allow for non-adjacent critical points

The termination of this process completes what is termed the X compaction. At this stage each column has a mask dimension associated with it in addition to the design rule tolerance to the adjacent column to the right. The same procedure is then repeated for the rows that comprise the cell. In addition to the backtracking step described above, an arc is swept out by the grid point in question to account for oblique design rule spacings. This is shown graphically in Figure 8. For instance during the X compaction, *wire1* may be moved arbitrarily close to *wire2*. The oblique checking causes *wire2* to be vertically spaced from *wire1* during the Y compaction to obey design rules. When the Y compaction is completed the spacings of rows and columns are stored in the *design grid* file shown in Figure 1.

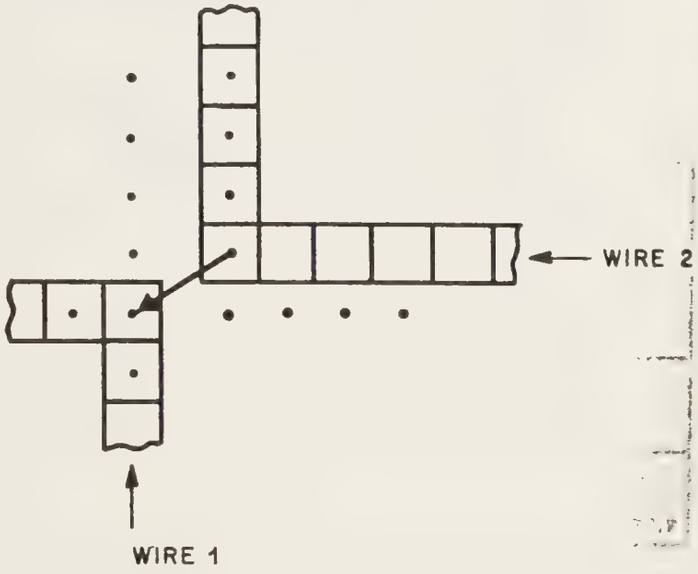


Figure 8. Oblique tolerance checking during Y compaction

It was mentioned that the critical compaction points were stored during compaction. These may now be used as an overlay on the original ICDL graphical representation to allow the designer to interactively change the topology. An important philosophical point emerges here. Previous compaction algorithms have included "jog" insertion as a method of changing the topology to improve compaction. This is not done in this system for a number of reasons-

— It increases the complexity of the compactor

— It introduces unforseeable topology changes

— Jogs form a minor role in space saving topology moves

The first point is minor, but of some import on a small machine. The second point is rather more serious in situations where cells have been designed to abut to complete a larger building block. In these instances, connection points which have been assigned along boundaries may be disrupted by jog insertion. Finally, perhaps the most important reason comes from the extensive experience gained using this system. It has been found that when a particular cell does not meet some predefined mask dimension, rather drastic topology alterations are usually necessary. This ranges from

re-routing a wire, to interchanging directions of the aluminum and polysilicon layers. Thus, with this in mind, it is a better strategy to allow the designer to interactively make *all* topology changes and then run a compactor, the output of which is predictable. This allows the designer to interact more smoothly with the algorithm. The MULGA system supports this style of design by providing compaction feedback to the designer and an efficient man-machine dialogue supported by the color graphics editor.

3.3 Performance

Figure 9 shows the relation between compaction time and cell size in grid units. The algorithm has essentially a linear execution time with respect to the area (in grid units) of a cell. Thus if the density of cells is comparable, this constitutes a linear dependence with respect to the number of elements. Approaches in [7] and [8] indicate complexity of N(1.5). Again it must be stressed that this performance greatly enhances the interactive design style.

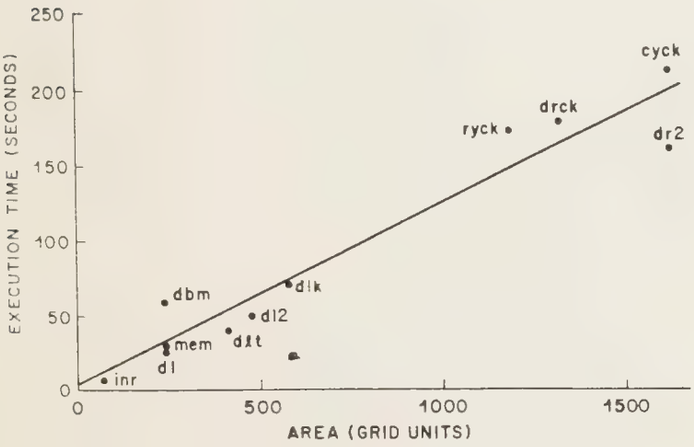


Figure 9

Virtual grid compaction algorithm performance

4. PHYSICAL MASK GENERATION

The MULGA system and the compaction strategy described may be used to design cells for various styles of chip design. For instance in a standard cell system, the advantage of using a symbolic approach is that physical design-rule independent cells may be constructed. This is especially advantageous as rapid technology advances force changes in processes and hence design rules. Another style of design that is supported is what amounts to custom design. However in contrast to other approaches, the use of symbology reduces the complexity of cell design to the point where a large number of performance verified cells may be constructed by an individual designer. To date, this is the style of design for which the system has been used, and this section illustrates how the construction of large cells is supported by the compaction scheme.

4.1 Structured Cell Composition

A designer starts the design of a chip or section thereof by first generating a *floor plan*. This takes into account the data and control flow of the network to be designed, and in addition partitions the circuit into manageable subcircuits. This may be repeated recursively until the designer decides the complexity of a sub-circuit is appropriate for design. Taking into account the global communication dictated by the floor-plan and the circuit itself, the cell is then designed. This is repeated for all cells at the bottom of the hierarchy. Cells are then combined by either abutment, repetition or simple routing. Once the top of the hierarchy is reached, the designer may determine the size of the layout and the critical portions of the design with respect to final mask size. Such critical areas may receive additional attention to improve packing density. Once this optimisation step has been completed a mask set may be generated.

The principle of defining performance verified cells and then combining them through the techniques described above has worked extremely well in practice. Once large functional blocks have been designed, conventional automatic techniques may be used to interconnect them[10].

4.2 Macro-Cell Construction

As a first step in constructing larger cells, which are pitch-matched to each other a simple language defines the relative placement of the cells. The following example is illustrative,

```
PLACE linty 0 0 a
NORTH a sl2y b
EAST a lintg c
NORTH c gl2 d
```

Translated, this means,

— Place cell *linty* at coordinate 0,0 and call this instance *a*

— North of *a* put cell *sl2y* and call it *b*

— East of *a* put cell *lintg* and call it *c*

— North of *c* put cell *gl2* and call it *d*

With the component cells shown in Figure 11a this leads to the placement shown in Figure 11b.

In addition one may then specify the nets that are to be connected (or pitch-matched) which are illustrated by the following examples,

```
a.vss.N  b.vss.S
a.yi.N b.si.S
a.cp1.W c.cp1.E
c.ld.N d.ld.S
b.a0.W d.a0.E
```

For example, the first command means,

Find all the North facing *vss* pins on *a* and match them to the South facing *vss* pins of *b*

A program checks for the correct matching of these specified connection points reporting any errors. Once this process has been completed the virtual grid points at which adjacent cells have to match in the physical domain have been defined.

Correct matching of virtual grid coordinates in the mask domain may be achieved in two ways. Firstly, the composite cell shown in Figure 10. may be split into "pseudo-cells" as shown in Figure 11. These have been designated *ab, ac, cd* and *bd*. Each of these cells is compacted. The mask file for cell *a* is then generated by using the X design grid file of *ab* and the Y design grid file of *ac*. This method is similarly applied to *b* (X- *ab* Y- *bd*), *c* (X- *cd* Y- *ac*) and *d* (X- *cd* Y- *bd*). By using this strategy all grid points on a common boundary match in the mask domain.

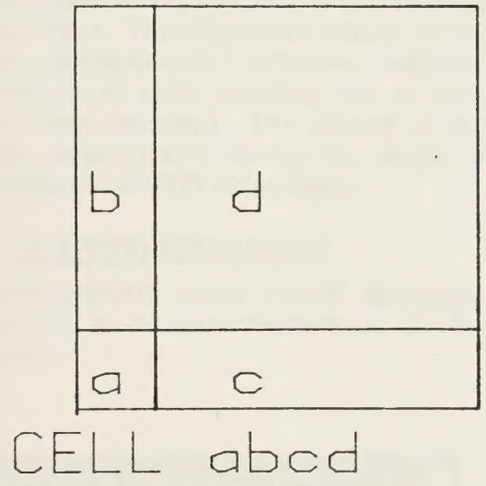


Figure 10. Composite cell abcd

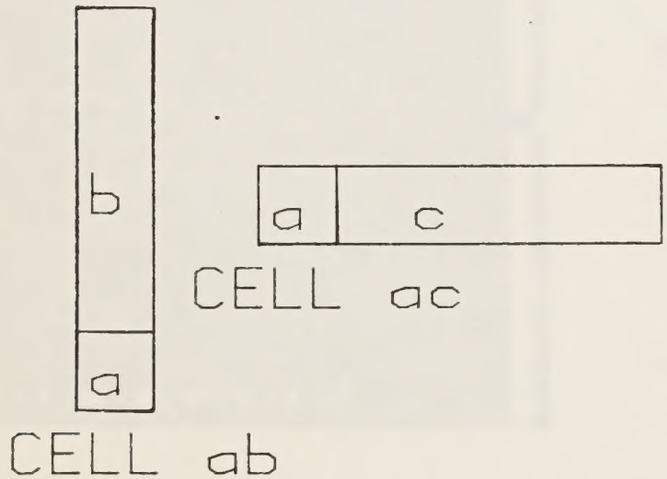


Figure 11 Pseudo cells used for pitch-matching

A refinement of this technique uses the connectivity specified previously. Here each cell to be considered is individually compacted. A program then examines the design grid files of each compacted cell at the boundaries and the grid points specified by the connection rules, and constructs a new file called a *mask coordinate* file which in effect combines the design grid files of the cells concerned according to the connection constraints.

Cells may be defined at the mask level only. These cells have fixed design grid files to which stretchable cells may be pitch matched. If this is impossible, either the variable cell is redesigned or a routing channel is employed. An example of such a cell would be the connection of a symbolically defined cell to an I/O pad.

Once the design grid and mask coordinate files have been adjusted to reflect to communication between adjacent cells, a program uses the original ICDL description and these files to generate XYMASK, the Bell System mask description language. Generation of other languages such as CIF is also possible. When mask conversion is complete the mask output may be viewed on the color graphics display. If changes are to be made in the layout, then the designer returns to the symbolic level.

5. EXPERIENCE

The compaction technique described in this paper has worked well with evolving layout methods that favor completing routing first followed by placement of devices and local interconnect[11], and the construction of generic bit-slice structures [12]. A requirement in these methodologies is a compaction algorithm that does not alter the topology of communication paths that are inherently dependent on the structure of a cell. The interactive nature of this system leads a designer to the rapid capture and subsequent modification of the topology. Using the system, a designer aquires a far more global view of his circuit at a far earlier stage than with more conventional methods. This allows for the efficient planning of effort on areas of the design which will yield the biggest payoff. In the experience gained so far with the system, one or two iterations seem to yeild an effective layout. The design rule free environment tends to encourage experimentation with the layout of complete structures prior to a decision being made about a final embodiment. For example, when constructing a bit-slice, a decision may be made to determine whether each cell should be long and thin or short and fat. In this case two layouts would be created and compared in area and aspect ratio to determine the most suitable for inclusion in the design.

It is important to note that although the system works well with more structured design styles, it is equally at ease dealing with less regular styles. The tendency in these situations is that the system guides a designer towards a more structured layout than might have been initially entered.

A chip photograph of the data path of a special purpose 16 bit CMOS processor is shown in Figure 12. This chip was designed on the MULGA system in 2 man-months. It contains 5000 transistors contained in approximately 30 cells. In addition to the CMOS chip two smaller NMOS chips have been designed and successfully fabricated. Currently, a 11000 transistor 16-bit CMOS special purpose processor is being fabricated.

6. SUMMARY

The concept of a virtual grid has been introduced as a means of describing and manipulating symbolic "stick" type layouts. A compaction algorithm that is a component of the interactive symbolic layout system that supports virtual grid layout has been described. It combines fast execution with predictability. The compaction method has been found to be very efficient in evolving structured layout styles. The interactive nature of the complete system compliments this efficiency. Additionally, inherent methods of pitch matching cells to build macro-cells have been described. The efficacy of this system has been demonstrated during the design of chips in both CMOS and NMOS technologies.

7. ACKNOWLEDGEMENTS

Bryan Ackland provided many fruitful discussions and ideas in addition to valuable feedback as the first "user" of this system.

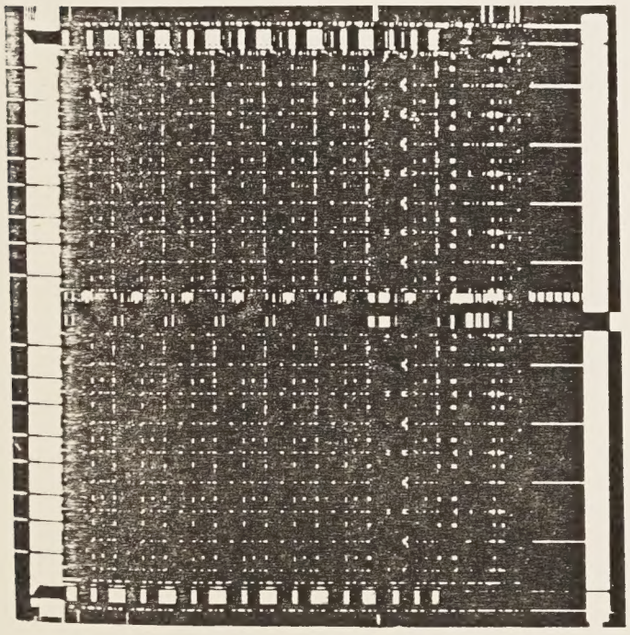


Figure 12. Chip photograph of 5000 transistor CMOS circuit designed with MULGA

REFERENCES

[1] Weste, N.,"MULGA- An Interactive Symbolic Layout System for the Design of Integrated Circuits," accepted for publication in the Bell System Technical Journal.

[2] Buchanan, I.,"Modelling and Verification in Structured Integrated Circuit Design," PhD Thesis,University of Edinburgh,Scotland, 1980.

[3] Ackland, B. and Weste, N.,"Functional Verification in an Interactive Symbolic IC Design Environment," Proceedings of the 2nd. Caltech Conference on VLSI, Pasadena, CA, 1981.

[4] Akers, S.B., Geyer, J.M. and Roberts, D.L.,"IC Mask Layout with a Single Conductor Layer," Proceedings 7th. Design Automation Workshop, San Francisco, 1970, pp. 7-16.

[5] Williams, J., "STICKS- A Graphical Compiler for High Level LSI Design," Proceedings of the 1978 NCC, May 1978, pp. 289-295.

[6] Dunlop, A., "SLIP: Symbolic Layout of Integrated Circuits with Compaction," Computer Aided Design, Vol. 10, No. 6, November 1978, pp. 387-391.

[7] Dunlop, A., "SLIM- The Translation of Symbolic Layouts into Mask Data," Proceedings of the 17th. Design Automation Conference, June 1980, pp. 595-602.

[8] Hsueh, M. Y. and Pederson, D. O., "Computer-Aided Layout of LSI Circuit Building Blocks," Proceedings of the 1979 International Symposium on Circuits and Systems, July 1979, pp. 474-477. (also University of California, Berkeley Memo No. UCB/ERL M79/80, 10 Dec, 1979 "Symbolic Layout and Compaction of Integrated Circuits)

[9] Losleben, P. and Thompson, K., "Topological Analysis for VLSI Circuits," Proceedings of the 16th Design Automation Conference, San Diego, 1979, pp. 461-473.

[10] Persky, G., Deutsch, D.N. and Schweikert, D.G., "LTX - A Minicomputer-Based System for Automated LSI Layout," Journal of Design Automation and Fault-Tolerant Computing, Vol. 1, No. 3 (May 1977), pp. 217-255.

[11] Lopez, A.D. and Law, H.F.,"A Dense Gate Matrix Layout Style for MOS LSI," IEEE Journal of Solid State Circuits, Vol. SC-15, No. 4, Aug. 1980, pp. 736-740.

[12] Johannsen, D., "Bristle- Blocks", Proceedings of the 16th Design Automation Conference, San Diego, 1979, pp. 310-313.